SuperGlue: Standardizing Workflow Components

Alexis Champsaur*, Jay Lofstead†, Jai Dayal*, Matthew Wolf*,
Greg Eisenhauer*, Ada Gavrilovska*
*School of Computer Science, Georgia Tech, †Sandia National Laboratories

***Abstract*—Scientific simulation workflows have traditionally used the storage array as a staging place between the components that make up the workflows and employed a high-level task manager to trigger these components. Recent work has been investigating moving these workflows to purely online models, thus avoiding the well-known performance bottleneck of storage arrays. One key advantage of moving to a purely online model is the ability to launch the entire workflow as part of a single submission, thus avoiding the challenges that existing workflow management systems face in launching individual components on HPC platforms. Still, there lacks a generic way to assemble complete workflows, even with many workflows using similar analytical routines.**

In this work, we present Superglue, an approach for assembling entire workflows from existing components and submitting workflows as single jobs, all through a single job submission shell script. We leverage the self-describing, process size- and placement-independent properties of data exchanged by the Adaptable I/O System (ADIOS) as well as the MxN publish-subscribe model of Flexpath to assemble and run three Superglue workflows based on three different and widely-used simulation codes exporting different data formats. We show that the performance cost of a componentized approach to assembling workflows is small, especially when considering the ability to assemble complete workflows "out of the box." While keeping in mind performance considerations, we present the building blocks as well as an approach to expand a library of such components, which we hope can eventually meet a wide variety of analytical needs in the scientific computing community, in the common scenario of submitting a workflow as a single job to a supercomputer scheduler.

I. Introduction

As extreme computing architectures continue to evolve, it is clear that I/O is an increasingly significant problem. Projections suggest no significant parallel file system I/O bandwidth growth through the next 100x increase in compute capabilities. The traditional HPC approach of writing complete checkpoints or analysis dumps for later processing to gain scientific insights is becoming infeasible. Even with technologies like Burst Buffers, limited capacity reduces the usefulness. As such, in situ and in transit analysis and visualization toolkits with the capability to reduce, process, and otherwise mitigate the raw increase in data volume and throughput requirements are increasingly important.

Scientific workflows, which capture a series of such analytical steps for a particular scientific discovery goal, usually by operating on large quantities of data, have also followed this trend of moving the analytical components online (i.e., executing simultaneously with the driving scientific code). A number of scientific workflow engines exist. However, while workflow engines such as Kepler [1] and DAGMan [2] offer flexible ways to assemble components with rich functionality to manage the control flow, they have not been universally adopted due to some of their limitations and complexity in the HPC environment.

An example illustrating the complexities comes from the Oak Ridge Leadership Computing Facility (OLCF). Kepler was used for several workflows for the fusion simulation users. While the initial goal was that an internal, expert resource would create the workflow, including the data transformation code required to allow downstream components to operate on the output of upstream ones (i.e., the "glue code"), the workflow should be able to be maintained easily by the application scientists. Unfortunately, the expert was required regularly as the configuration evolved and scaled. The complexities of maintaining the components and the glue code as the output shifted and managing the deployment was too high. Even then, the workflow operated offline.

One major limitation to existing workflow engines is that given only a simulation code, a scientist cannot easily assemble a workflow "out of the box." Indeed, even with the re-use of code that performs common analytical procedures and with the automation of control flow allowed by existing engines, for a scientist to assemble and launch a workflow using existing tools, significant effort is required to (a) allow the existing analytics code to operate on the specific data formats used in this particular workflow, (b) reconcile the discrepancy between the potentially differing levels of parallelism of the various components, and (c) map the workflow components to the available resources, which in the HPC space is usually a supercomputer.

Proposed infrastructure for online workflows is less mature but can offer the necessary functionality for assembling workflows ad-hoc [3], [4], [5], [6]. For example, in PreDatA [3], separate applications are run to offer different resilience domains, but the actual placement of operators may be different depending on the performance characteristics requiring coding changes for each placement decision. DataSpaces [4] offers a way to map data from one application to another, but it has no mechanisms to manage what is a complete data set, when it begins in an output stream, and when it ends. The user has to determine if what is being requested is complete and correct or not. Data Staging [7] and in particular data staging with processing capabilities [6], [8], are a solid step forward in providing online functionality for workflows, but do not offer a generic approach for their assembly. SmartPointer [5] and DataStager [6] both offer examples of integrated workflows, but the construction is completely custom and not portable to

other workflows.

This lack of a robust infrastructure and the difficulty in using these tools directly to assemble an HPC workflow out of the box based on an existing scientific code, simulation or other, and launching it in a way that is as simple as a single job submission have limited their adoption.

Newer work is extending these systems to provide more generic functionality in workflow assembly and management [9], [10], [11], [12], [13], [14], [15]. In particular, FlexPath [9] is a publish/subscribe system that allows two concurrently running MPI executables to exchange data that is globally partitioned among their ranks, even when the number of writers and readers differ. It is implemented as a transport method for the Adaptable I/O System (ADIOS) [16], which provides a unified interface to parallel I/O over a rich variety of underlying data encoding, transport, and placement technologies. With ADIOS, FlexPath provides a simple interface for asynchronous, MxN communication between simulation and analytical components through the abstraction of a stream.

In this work, we leverage the stream-based, MxN I/O interface offered by ADIOS and FlexPath to present SuperGlue, a collection of generic workflow components that allow for the easy assembly and deployment of a variety of scientific workflows.

SuperGlue allows workflows to be assembled by using existing, generic data transformation and analysis components which do not require any code changes for specific workflows; the only code changes required when using SuperGlue are for redirecting of the output of interest in the driving simulation code through ADIOS.

SuperGlue components solve a number of the aforementioned problems in current online workflow-oriented technologies. First, they leverage the self-describing property of data exchanged through ADIOS to allow for operation on data having very different formats. In scientific workflows, the data of interest is generally composed of large, multi-dimensional arrays; in SuperGlue, this translates to the ability to operate, as much as possible, on data having any number of dimensions. To address the difficulty and complexity in mapping workflows to supercomputers using existing workflow technology, which is targeted at a variety of resources outside of HPC, such as Grid and Web resources, SuperGlue reduces the launch of an entire workflow to the submission of a single job script. Finally, it uses the MxN publish/subscribe ability of FlexPath, as well as consistent semantics describing the data that are passed through the components, to exchange data while preserving semantic integrity even in the presence of varying levels of parallelism at different stages of the workflow.

Rather than providing a complete collection of such components, we provide here the basic building blocks of what we hope will be a library of components that can eventually be used to assemble a rich variety of workflows, driven by simulation codes exporting a variety of different data formats.

To demonstrate the functionality of the SuperGlue approach, we deploy three SuperGlue workflows based on three different scientific simulations, namely LAMMPS [17], GTCP [18], and GROMACS [19]. We show that SuperGlue allows workflows to scale well and that simple experiments allow for the selection of appropriate levels of parallelism at different stages of the workflow.

One disadvantage that might be perceived in the fine-grain componentization of a workflow that naturally comes with the use of SuperGlue is the potential loss in performance. However, we show through a performance comparison of a componentized, generic workflow with a corresponding ad-hoc workflow which uses a monolithic analytical component that the componentization of analytical routines does not lead to a noticeable change in the overall performance of the workflow.

The rest of the paper is organized as follows. First is a survey or related work in §II. Second, in §III, is a discussion of the three workflows we use to drive our insights. Next is an overview of the concepts behind SuperGlue in §IV. In §V, we provide an overview of the reusable components created. In §VI, we discuss the challenges of integrating reusable components. An evaluation is presented next in §VII. Finally, we present a discussion of Conclusions and Future Work in §VIII.

II. Related Work

Existing workflow systems have typically only been able to offer generic, reusable components when the workflow system is for a particular niche with a fixed datatype and standardized interfaces. For example, enterprise document processing systems may all work against a single database with each user only seeing their current worklist. As documents are processed, they are moved to the next work queue, or completed state, according to hand-coded rules [20]. The Workflow Management Coalition [21] has developed standards to make enterprise process workflows more portable. These standards are not intended to make components reusable, but to make different workflow engines able to inter-operate or to port a workflow from one engine to another. The actual communication interfaces and data types are left to the components themselves.

More directly related to this work and the scientific community are workflow engines and frameworks custom-made for the parallel computing environment. Pegasus [22] and DAGMan [2] work together to offer an engine to execute a workflow and a front-end system to construct the workflow process itself. However, the focus in DAGMan is on specifying dependencies between the jobs involved in the workflow, so as to execute components only when required and provide resilience. In contrast, in SuperGlue, the entire workflow is executed at once, with FlexPath allowing readers and writers to block until the corresponding writer or reader is available for data exchange. Furthermore, the Pegasus/DAGMan engine does not provide a library of generic, reusable components.

As an alternative to DAGMan, Swift [23] is a scripting language that allows ordinary applications to be composed into parallel scripts, and eventually into workflows, with dependencies specified in the script. However, these applications themselves must be written outside of the Swift script.

Kepler [1] offers a nice GUI for assembling different kinds of scientific workflows. It offers different directors to manage how the workflow will be executed. Each component is an actor with channels connecting actors. As mentioned in the Introduction, complex workflows using Kepler have been assembled for many communities, including fusion science. However, the large collection of components that come with Kepler are mainly designed to work in a single Java Virtual Machine instance; using HPC-scale components requires significant effort both in coding the custom components and in allowing the I/O methods used to be understood by the higher-level Kepler engine.

To address much of the complexity of communicating between separate parallel components, inline approaches are being investigated. We use "inline" in the sense of "linked into the same process." Catalyst [24] offers a way to integrate the ParaView [25] analysis and visualization system directly into the simulation executable. Catalyst strips out many features to reduce the memory footprint and then requires explicit calls from the host application into Catalyst routines with predictable data types on in-memory data structures. While this can work for limited kinds of data processing, it clearly cannot take advantage of additional resources available for off-site analysis, instead taking cycles away from the main scientific code.

Libsim [26] has a similar relationship to VisIt [27] as Catalyst has to ParaView. Both Libsim and Catalyst have a strict limitation that offline workflows do not. In both cases, because they are running on the same nodes as the simulation, time series analysis and visualization can be difficult or impossible. The potential scaling impact on the simulation because of limitations in Libsim or Catalyst can prevent their use for the most important use cases at extreme scale.

A middle path between in situ and offline processing was investigated in PreDatA [3]. This work demonstrates that the placement of the analytics can significantly affect the performance of workflows, and that this placement can be determined in part by the communication characteristics of the analytics components.

Companion work to SuperGlue within this same project is Bredala [28]. This work presents an attempt to build a data model for in situ workflows. It has some similarity to FFS [29]. Unlike FFS, which is part of a much more complex infrastructure for typed messaging between distributed processes, Bredala strictly focuses on a data model that can preserve semantic integrity across redistributions.

Overall, while all of these efforts are addressing different portions of the online workflows puzzle, none of them address the idea of general, reusable data transformation and analysis components for the assembly of entire HPC workflows out of the box.

III. Workflows

We designed and implemented three realistic online workflows based on scientific codes having large user bases: the LAMMPS Newtonian particle simulator [17], GTCP, a particle-in-cell Tokamak simulator [18], and GROMACS, a biomolecular dynamics code [19]. While all these workflows eventually turn part of the simulation data into histograms of certain quantities of interest, there is significant variation in how they arrive at their final result. Creating similar types of results, and this using some of the same components but in significantly different ways, has allowed us to gain important insight into how best to design components that can be used to build a wide variety of workflows. We first describe the workflows from a general point of view and then the components in greater detail, illustrated in Figure 3 Figure 4 and Figure 5.

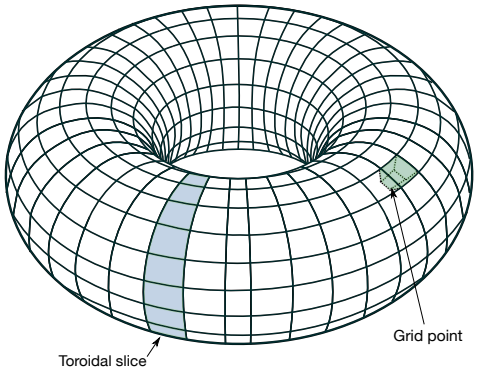

Fig. 1. Representation of GTCP Toroid, modification of [30]

A. LAMMPS Workflow

We configure LAMMPS to simulate a disruption (a "crack") in a thin layer of particles and output 5 numerical properties describing each particle in the simulation at regular timestep intervals. This corresponds to two-dimensional data (particles as one dimension and properties of interest as another) and among these properties are the three-dimensional components of the particles' velocities. The workflow outputs one histogram per timestep showing the distribution of the magnitudes of the particles' velocities for all particles involved in simulation at that point in time.

B. GTCP Workflow

GTCP, a code that simulates a toroidally confined plasma, splits the solid into toroidal slices, each made up of a number of grid points. For each of these grid points, it outputs 7 properties of the plasma such as pressure and energy flux. This division of the toroid is illustrated in Figure 1. The output of the simulation is therefore a three-dimensional array in which the dimensions span: (a) toroidal ranks (toroidal slice number), (b) grid point numbers, and (c) various properties that describe each grid point. In this workflow, the per-timestep histogram generated shows the distribution of per-gridpoint perpendicular pressures across the entire simulation.

C. GROMACS Workflow

Among other quantities, GROMACS outputs the three-dimensional coordinates of the atoms involved in the simulation at regular intervals. The data array itself is two-dimensional: 3D coordinates over all atoms. From these, we obtain a histogram of the distances of the atoms from the

origin for each timestep, showing an evolution of the spread of the particles through the simulation in a human-readable way.

D. Discussion

All three workflows are built using only the simulation code, modified to allow for ADIOS output, as well as SuperGlue components, unchanged when the same component is used in different workflows. As we will explain in more detail later, the components' ability to fit into different workflows is achieved using command-line parameters specified in the batch job submission script used for launching the whole workflow on the target cluster.

All three workflows create histograms as their final result, as histograms are a common final, human-readable result of many types of workflow. However, the components presented in this paper are meant only to illustrate the building blocks of a library of such tools, of which histogram is only an example.

Also, the driving simulations generate very different data, so selecting the relevant data from the simulations and formatting it in a such a way that the final Histogram component can operate on it is done differently in each workflow.

Much of the challenge is the data selection and mathematical manipulation to obtain the quantities of interest in a way that (a) presents an intuitive interface to the scientist constructing the workflow and (b) is useful in different types of workflows in which data have different sizes and dimensions. This difficulty arises because (a) the data at each stage of the workflow is distributed over the collection of processes, with varying levels of parallelism at each stage, and (b) the extraction and re-arrangement of multi-dimensional, distributed data, in a way that is configurable by the user at runtime is challenging.

Addressing these challenges has offered us some insight in the design of such generic tools.

IV. Design

In this section, we first present some insights in the design of generic workflow components, and we then discuss some of our implementation choices and how they allowed us to build these components.

A. Insights: Overview

By evaluating the presented workflows and considering other workflows with which the authors are familiar, four key insights are revealed.

1) To allow for the greatest variety of workflows, data manipulation primitives and data analysis components should be packaged in similar ways – that is, regardless of their individual complexity, the pieces that make up these workflows should export compatible interfaces as much as possible.

2) The ability to handle multi-dimensional data, along with the consistent labeling of dimensions and quantities as metadata, allows for components that are highly adaptable and simple to use.

3) While different types of components understand varying levels of semantics, maintaining a high level of semantics (i.e., labeling quantities and dimensions as much as possible) early on and when passing through components that do not necessarily require all of these labels allows for the most functionality downstream.

4) Because programming languages understand multi-dimensional data as being in a specific order in memory, there is a need for glue components that re-arrange data and re-label its dimensions without necessarily changing its size. Indeed, when data is stored in a database on disk, it is simple to gain a desired view of the data, for example by using SQL. However, in the middle of a real-time workflow, data must be presented to the components in a format that they expect and understand. This requires a specific ordering of data in memory. We expand on this topic in §IV-D.

These insights guide the design for the reusable glue and analysis components presented in this paper. From a general perspective, designing a smaller number of components to assemble workflows with finer step decomposition allows for more general processing than designing more numerous components each having more complex functionality. And, as we show in §VII, componentizing the analysis involves only minor changes in overall workflow performance.

B. Multi-Dimensional Data Support

Many scientific codes serialize their output, effectively packing multi-dimensional data into a single dimension. However, this technique offers little information on the data to downstream components in a workflow.

For example, two of the simulations used in the workflows presented in this work inherently pack their output into one-dimensional arrays for native output, even though LAMMPS' output is logically two-dimensional and that of GTCP three-dimensional. In order to use the same glue code to extract desired data from both outputs, this code must be able to operate on two-dimensional data as well as three-dimensional data. In fact, if the glue code is able to operate on any number of dimensions, all that must be provided to it at runtime is the dimensions and indices to extract. This is precisely the role of our *Select* component.

In general, it is advantageous to (1) design components that can operate on multi-dimensional data as much as possible and (2) format the output data of scientific applications as having well-defined dimensions. Emphasizing the support for multi-dimensional data in the design of workflow components allows for maximum compatibility between the interfaces of components by providing a consistent way to refer to the data.

Still, while multi-dimensional data support provides a consistent way to refer to the data, not all components should be designed so as to work with any number of dimensions. For example, we found it advantageous to design our *Histogram* component to work with only one dimension since this is a natural input for a histogram operation.

C. Semantics

When data is organized under clearly defined dimensions, labeling these dimensions and the indices they hold as the data

goes through each component lets downstream subscribers refer to dimensions using their names. Because the data is potentially re-sized and re-arranged in the course of a workflow, it is useful to maintain such semantics as much as possible. However, the absence of labels should not block the workflow execution.

For example, in both the LAMMPS and the GTCP workflows, we modified the simulations so that they label the quantities that they output in the dimension that holds these quantities. This lets the Select component extract the quantities of interest by referring to them by name. However, we did not label the dimensions themselves, rather referring to them by number.

When they exist, these labels are concatenated into a header string passed along to the next component in the workflow as another variable through the ADIOS interface, alongside the main simulation data. While in our current implementation, Select is hard-coded to always looks for such a header describing the quantities in the dimension of interest, we can easily extend this functionality to let the user either refer to dimensions and indices by name, when headers exist, or by number, when they do not.

In general, labels should be used as much as possible, but they should also be kept optional.

D. Dimension Reduction

As discussed earlier, there is a need in scientific workflows with fine step granularity for glue components that re-arrange and re-label data without necessarily changing its size. This is illustrated by the need for the *Dim-Reduce* operation in the GTCP workflow.

In its raw output, GTCP keeps track of the toroidal slice that produces the data of interest by using a dimension that spans the "Toroidal rank" of grid points. In our workflow, we wish to create histograms encompassing all grid points in the toroid, thereby eliminating the concept of "Toroidal rank" along the way and instead growing the dimension that spans the number of grid points.

Programming languages represent multi-dimensional arrays in a specific order in memory, so we cannot simply keep the data ordered as it is, change the number of dimensions and their sizes as ADIOS understands them, and assume that the new dimensions correctly reference the data.

The dimensions of GTCP's raw output are $N_0 \times N_1 \times N_2$, where:

- N_0 is the size of the *toroidal rank* dimension D_0
- N_1 is the size of the *gridpoint* dimension D_1
- N_2 is the size of the *property* dimension D_2

At one stage of the GTCP workflow, we wish to eliminate the concept of *toroidal rank* of the data. This is to bring the data one step closer to a format that Histogram understands: one-dimensional data. For the data to be one-dimensional, two instances of the Dim-Reduce operation are required. We focus here on the first of the two.

The desired result of this operation is that:

- The concept of *toroidal rank* of a particle disappears.
- The concept of *gridpoint number* loses its original meaning and takes on a new one; the indices in this dimension now cover all gridpoints in the toroid, rather than only those in a particular toroidal slice.
- The concept of *property* keeps its original meaning, and the size of its dimension is unchanged.

We can say that D_1, the *gridpoint* dimension, ***absorbs*** D_0, the *toroidal rank* dimension. The array resulting from this operation has dimensions $N_1' \times N_2$, where:

- $N_1' = N_0 \times N_1$ is the new *gridpoint* dimension D_1' size
- $N_2' = N_2$ is the *property* dimension D_2' size

We call this operation *dimension reduction*. Even though it does not change the size of the data set, we have seen in our implementation that it often changes the in-memory ordering of data. Consequently, it is still potentially a compute-intensive operation with sizable communication overhead when the dataset is large and many processes are involved in it.

This operation fits well into a SuperGlue component, and it is precisely the role of the Dim-Reduce component, which can operate on a dataset having any number of dimensions.

E. Implementation Artifacts

Though we refer to the SuperGlue components as single entities, they are distributed codes able to split computation on large dataset among the processes of which they are composed. More precisely, they are MPI executables written in C where all processes in a component belong to the same MPI communicator.

To connect components, we use ADIOS [16] for the flexibility in writing destinations, reading sources, and offering a typed data stream. In particular, we use the FlexPath transport. It implements a *stream-based* data exchange abstracted to the components through the ADIOS interface, is asynchronous, and allows for data exchange between any number of writers and readers. Therefore:

1. We can launch components of the workflow in any order: downstream components will wait for the availability of data from upstream components and upstream components will buffer data up to a certain size until they are able to send it downstream. In our experiments, we launch all components simultaneously.

2. Even if the number of processes used for one component is different from that used for the previous one in the workflow, each component can split the data (and therefore the computation) evenly among its processes.

3. A FlexPath stream actually corresponds to the internal buffering of data on the writer side until readers are ready to request the data, at which point the data exchange on the writer side is carried out by a separate thread. This asynchronous communication allows a SuperGlue component to move on to the computation of a subsequent timestep even when a downstream component is not yet ready to accept data, effectively overlapping computation and IO and offering high performance to a componentized workflow.

ADIOS and its transports, such as FlexPath, keep track of the data dimensions and their sizes. Therefore, when a

```
1  aprun -n 64 histogram velos.fp 16 velocities &
2  aprun -n 256 magnitude lmpselect.fp \
3    lmpsel velos.fp velocities &
4  aprun -n 256 select dump.custom.fp \
5    atoms 1 lmpselect.fp lmpsel vx vy vz &
6  aprun -n 1024 lmp_titan < in.cracksm &
7  wait
```

Fig. 2. SuperGlue example launch script, LAMMPS workflow

component receives a multi-dimensional array, it can discover the dimensions of the data and their sizes as defined by the previous component in the workflow and divide the global array and split the computation accordingly.

There is no need to re-compile SuperGlue components when using them in different workflows. Any configuration of a component for a particular workflow can be provided through run-time arguments. We will show more clearly how this is done in the next section, but in general, the user must specify the stream and array names from which to read and to which to write, and usually also dimensions and/or indices to operate on. Referring to streams and arrays using names allows users to easily chain together these components into potentially complex workflows.

V. Reusable Components

This section provides greater details about the individual SuperGlue components and how a small number of parameters allow them to operate on a variety of different input data formats in a user-specified way.

A. Select

Given an input stream that includes an array with any number of dimensions, Select extracts certain indices from one of the dimensions. Thus, it outputs an array with the same number of dimensions, but with the dimension of interest having a smaller size. In order to select the quantities of interest, the component uses a header which must be passed by the previous component in the workflow. The header is a list of strings that name the quantities in the dimension of interest. This allows for easy selection of quantities at runtime when Select is launched. For example, in the LAMMPS workflow, the simulation outputs the ID, Type, v_x, v_y, and v_z of each particle, where v_x, v_y, and v_z are the three-dimensional components of the velocity of a particle. Select discards the ID and Type, building a new array consisting only of the velocity components. The user must pass to this component the index of the dimension from which to select, as well as the indices of the quantities of interest within that dimension. Figure 2 shows an example launch script for the LAMMPS workflow. Lines 4 and 5 show how Select is launched. `dump.custom.bp` is the FlexPath stream written by LAMMPS; `atoms` is the array of atoms written by the simulation; 1 is the dimension number that spans the 5 atom attributes; `lmpselect.fp` is the stream that Select will write; `lmpsel` is the array that Select will write; `vx vy` and `vz` are the names of the quantities to be extracted, which will be found in the header written by LAMMPS.

B. Dim-Reduce

Dim-Reduce is a glue component that removes one dimension from its input array, "absorbing" it into another dimension without modifying the total size of the data. The other dimensions are left unchanged. This component can work with an input array having any number of dimensions. The output is an array with one dimension removed and with another dimension that has been re-defined. When using this component, the user must specify which dimension to eliminate and which to grow.

C. Magnitude

In our current implementation, Magnitude expects a two-dimensional array as input, where one dimension spans the data points at each time step (particles in the case of LAMMPS and grid points in the case of GTC) and the other dimension spans any number of components of the same vector, for example the three-dimensional components of velocity in the LAMMPS workflow. Magnitude calculates the magnitudes of the vectors from the values of their individual components and outputs a one-dimensional array of the new values. Which dimension is which in the input array is specified by the user at runtime. A small number of changes and a few start-up parameters could generalize this code to perform any number of common operations that calculate a quantity from many, applying a known formula over a two-dimensional dataset, thus allowing this component to be extended into a variety of others.

D. Histogram

The processes that make up the Histogram component partition among themselves a one-dimensional array of data. They communicate to discover the global minimum and maximum values in the array, create a number of bins between these two extremes, and then communicate again to count the number of values in the globally partitioned array that fall in each bin. The number of bins to use must be passed to the component when it is launched.

In our current implementation, one of the processes of Histogram writes the output to a file on disk. We chose this approach because this component is often used as an endpoint in the workflow and because the output of this component is generally very small compared to its input and can be easily written by a single process. However, letting this component output its data in the same way as the other components, that is as an ADIOS stream, and instead writing to disk when needed using a component specifically designed for this purpose would provide greater flexibility.

E. Implementation Details

The components are implemented as MPI executables written in C, varying in length from 191 lines of code (Histogram) to 459 (Select). Their code is publicly available in [31].

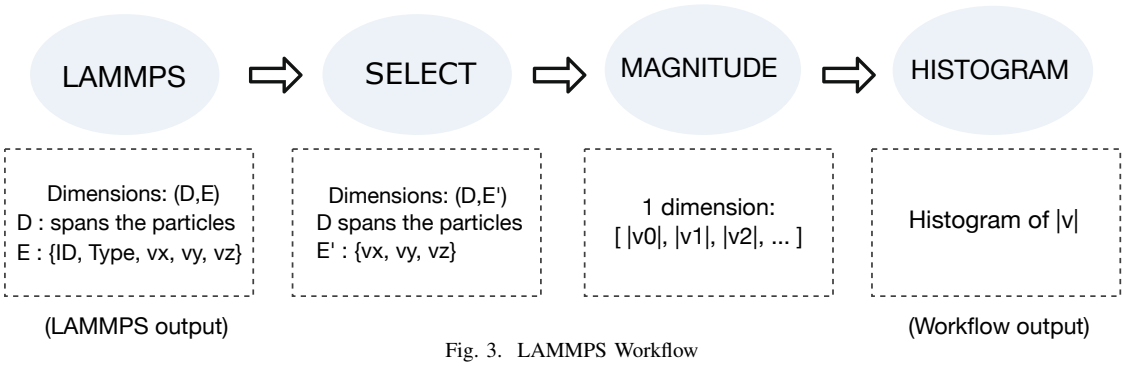

Fig. 3. LAMMPS Workflow

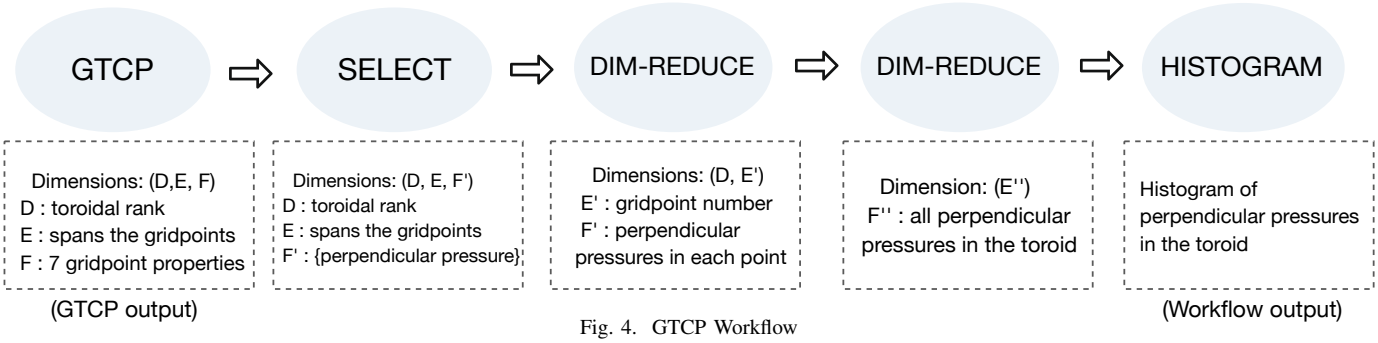

Fig. 4. GTCP Workflow

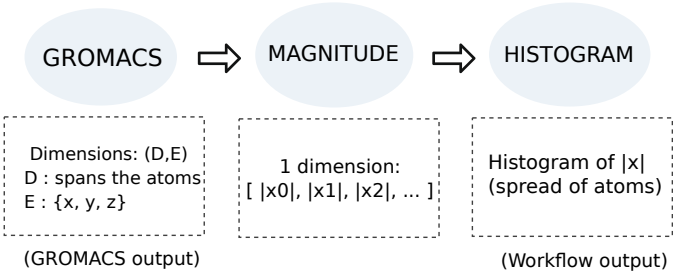

Fig. 5. GROMACS Workflow

VI. Integrating Reusable Components

To allow the simulation codes used to work with SuperGlue, we had to modify their output routines to use ADIOS. ADIOS expects multi-dimensional data to be packed as a linear array. LAMMPS and GTCP do this packing natively, but we had to add it to GROMACS. Roughly 70 lines of code are required to allow each simulation to work with SuperGlue, along with an approximately 25-line XML file that describes the data to ADIOS at run time. The code changes made to GROMACS version 4.6.7, as well as the workflow launch files, including the XML configuration file, can be seen in the public SuperGlue repository [31].

In general, in order to work with SuperGlue components, simulations have to specify through ADIOS the logical dimensions of their data and optionally create corresponding headers to label them and their indices, before writing the data itself.

A. Demonstrating in the Workflows

We built the LAMMPS workflow, illustrated in Figure 3, using only LAMMPS and SuperGlue components. We annotate the figure with details about how the data is manipulated at each step.

Data arrives from LAMMPS at the first SuperGlue component, Select, which extracts the velocity components from the raw output of the simulation. From Select, data is sent to Magnitude, which computes the magnitudes of the velocities. Magnitude outputs one-dimensional data, an array of the magnitudes it calculates, to the final component, Histogram, which expects one-dimensional data as input. The end result of this workflow is a series of histograms of the total velocities of the particles. There is one histogram created at each timestep at which the simulation would normally dump its data to disk.

The GTCP workflow is illustrated in Figure 4, and it too was assembled only from the simulation and SuperGlue components. Note that the workflows primarily differ in the data formats output by the simulations.

From GTCP, three-dimensional data first arrives at an instance of Select, which extracts one quantity of interest out of the 7 properties that describe each gridpoint. This quantity was arbitrarily chosen as the "perpendicular pressure," or pressure of the plasma perpendicular to the flow in the grid point of interest. Even if it contains only perpendicular pressures, the output of Select is still three-dimensional since this component maintains the original dimensions of its input. Because the Histogram component expects one-dimensional input, we first send the output of Select through two instances of our Dim-Reduce component, each of which eliminates a single dimension of the array without changing its total size. The final component, Histogram, outputs a histogram of the perpendicular pressures of all grid points at each timestep at which the simulation would normally output its data to disk.

The GROMACS output contains only the position vectors of the atoms. Therefore, it can be sent directly to Magnitude, which calculates the magnitudes of the distances of the atoms

TABLE I
LAMMPS SUPERGLUE VS. ALL-IN-ONE COMPARISON

SIM output	AIO time (sec)	Superglue time (sec)	LMP only (sec)
20 MB	115.26	116.51	115.03
80 MB	148.70	149.80	146.97
320 MB	154.65	157.65	153.69
1280 MB	155.32	157.98	152.48
5120 MB	167.39	168.79	165.22

from the origin, and Histogram then shows a distribution of these distances, giving an idea of the spread of the particles at each timestep of interest.

The four components used to build these workflows are only meant to illustrate an approach in designing a library of such components to allow for the assembly of a variety of workflows; they are not meant to satisfy most analytical needs.

In the repository, we provide a non-compilable template component (`template.c` in [31]), documented to show how to write other analytical components using the SuperGlue approach.

Still, that three different workflows based on very different scientific codes were built using only 4 components shows the potential of the approach.

VII. EVALUATION

In this section we use performance measurements on two of these workflows to show (a) that a componentized method in building workflows, which is inherent in the SuperGlue approach, is valid from a performance point of view, and (b) to show some of the scaling characteristics of the components. These results also serve to show that SuperGlue workflows were successfully deployed at various scales.

The evaluation is performed on Titan, the Cray XK7 machine at Oak Ridge National Laboratory. It consists of 18,688 nodes each with 1 16-core AMD Opteron CPU and 32 GB of RAM. The interconnect is a Gemini network. There is an attached Nvidia Kepler K20X GPU with an additional 6 GB of memory on every node.

A. Evaluation of Componentization

It is expected that assembling a workflow using generic components involves the use of finer-grain components than when using ad-hoc analytical routines specifically coded for a workflow of interest. One problem that might be anticipated in more componentized workflows is a decrease in overall performance caused by (a) more stages that require the coordination of readers and writers and (b) more points of actual data transfer.

To investigate this potential problem, we wrote a custom, all-in-one (AIO) component that performs the same analytical procedure as all the components involved in the LAMMPS workflow (outside of the simulation itself, however). We measured the start-to-end completion times of the two workflows at different scales.

Table I shows the start-to-end completion times at increasing scales, of (a) LAMMPS with the AIO component in the second column (b) LAMMPS with the full SuperGlue workflow in the 3rd column and (c) the LAMMPS simulation only with the output routines removed from the code in the last column. The measurements in the last column are meant to give an idea of the portion of the workflow completion time taken up by the simulation computation only.

A weak scaling approach is used, with approximately the same per-process data size throughout the scaling. The time is measured from the start of the simulation to the point when the last histogram of the last timestep is written. For each SuperGlue workflow run, the corresponding AIO workflow run allocates the same number of processes to the AIO component as the SuperGlue workflow allocates to the Select component. In the SuperGlue workflow, additional processes are allocated to the other two components (Magnitude and Histogram).

The results show only a small increase in workflow completion time for the SuperGlue workflows (with a maximum increase of 1.9%). As mentioned previously, the componentized approach involves more data exchange points in the workflow, thus leading to overhead from increased MxN coordination and data exchange. However, FlexPath allows for asynchronous data transfer. More specifically, when a writer writes to a "stream," it places the data in an internal buffer until the readers are ready to request it, at which point a separate thread in the writer handles the actual transfer. This overlap of computation and I/O amortizes the aforementioned overhead.

Also, in the SuperGlue workflow, the componentization of the analysis allows for a higher level of parallelism in the overall analysis. As an example, in the AIO component, the Select stage of one timestep only begins once the Magnitude stage of the previous timestep completes, since they exist in the same processes. In the SuperGlue workflow, these computational stages can run simultaneously.

Granted, more resources are allocated to the SuperGlue workflow runs to allow the additional components to execute. Also, much of the start-to-end time is spent on the simulation's computation. However, the measurements in the last column of Table I only give an idea of the proportion of overall time occupied by the simulation only, since when workflows are running, there is much overlap in the computation and I/O between the simulation and other components. All in all, this is an illustrative comparison that shows the validity, from a performance angle, of the componentized approach to building workflows.

B. Scaling

We carried out a number of experiments to determine some of the strong and weak scaling characteristics of the components.

Strong scaling measurements are useful for determining appropriate process sizes for the components, based on the size of the data set they operate on. For these strong scaling measurements, we varied the process size of a single component at a time while fixing that of the other components involved in the workflow, all the while using a fixed output size from the driving simulation. We timed the completion of a whole timestep taken arbitrarily in the middle of the workflow

TABLE II
GTCP EVALUATION CONFIGURATION SETTINGS

Component Test	GTCP Procs	Select Procs	Dim-Reduce 1	Dim-Reduce 2	Histogram Procs
Select	64	x	4	4	4
Dim-Reduce 1	128	32	x	16	16
Dim-Reduce 2	128	32	16	x	16

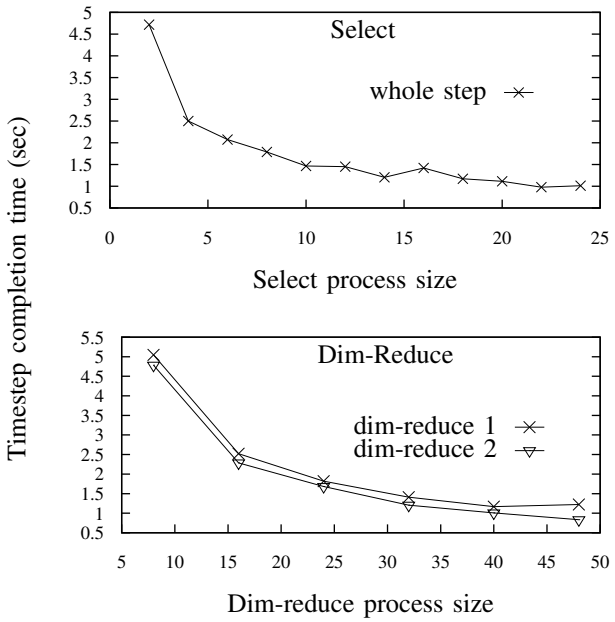

Fig. 6. SuperGlue strong scaling in the GTCP workflow. Whole timestep completion time (secs) for Select and both instances of Dim-Reduce used in the workflow are plotted against process size

of one component of interest at a time, taking an average over all processes involved in that component's timestep.

In Figure 6 we show some results from the GTCP workflow. The corresponding configurations (process sizes for all components) that were used to obtain these results are listed in Table II. These strong scaling results show a general trend seen in the strong scaling experiments for SuperGlue components: a linear domain of scalability followed by a turning point and eventual flattening of the curve. This shows that given a particular workload, users can select SuperGlue process sizes that match their resources and performance requirements.

Similar measurements exist for the LAMMPS and GROMACS workflows. These are available in the repository [31].

The third column of Table I already shows promising weak scaling characteristics of SuperGlue in the LAMMPS workflow. That is, there is little difference in the end-to-end completion time of the workflow for simulation output sizes varying from 80 MB to 5120 MB (a factor of 64 increase in simulation output size). For the 5120 MB run, the configuration used was 1024 LAMMPS processes, 256 Select processes, 256 Magnitude processes, and 64 Histogram processes.

Per-component, per-timestep results showing good weak scaling characteristics of SuperGlue also exist for the GTCP workflow and are also available in [31]. Space limitations prevent us from showing them here.

VIII. CONCLUSIONS AND FUTURE WORK

This paper presents SuperGlue, a demonstration of making generic, reusable components for scientific simulations. By decomposing the operations into small chunks, we achieve components that can be reused, without modification, for a variety of different workflows. In this work, we investigate using a stream-based structure with generic components to achieve easier to build and use workflows. Stream-based, generic workflow components should be designed to allow the greatest variety in their arrangement and for a maximum number of downstream subscribers. Designing components with the ability to handle data having any number of dimensions provides a very useful way to link them together. Maintaining a high level of semantics upstream, for example by labeling dimensions and certain quantities inside of these dimensions, gives a good understanding of the data to downstream components. There is a need for components that organize the data in a format that downstream components can understand.

Through the demonstration of generating a velocity histogram for LAMMPS, a pressure histogram for GTCP, and a distribution of the spread of the atoms for a GROMACS experiment, we demonstrate reusing the same components over different data formats and application types.

While this work leverages ADIOS and the FlexPath transport, this is not the only approach for addressing reusable components. Other, similar approaches can also work well. However, in this case, the data annotation provided by this connection infrastructure helps enable reusable components by offering necessary metadata to perform general operations.

Future work involves expanding the generic components library to include a variety of analytical operations. In particular, the SuperGlue components presented in this paper result in an output dataset having either the same size or a smaller size as the input. Analytical procedures that lead to an increase in data size, such as all-pairs calculations, are common and can be implemented using the SuperGlue approach.

Also common are operations that require the in-memory buffering of a dataset throughout a series of timesteps. An example of this is the calculation of root-mean-square deviations of atomic positions from a fixed, starting set of positions.

Two current limitations of SuperGlue are that all components are launched at once and workflows cannot branch.

To turn SuperGlue into a true Workflow Management System, we hope to leverage ADIOS' ability to have several "write groups" so as to allow for the development of a *Fork* component that would permit the creation of workflows described by directed acyclic graphs. And, to manage the

execution of workflows over longer periods of time, we plan on investigating the incorporation of SuperGlue into higher-level workflow management systems such as Kepler and DAGMan.

The components we have developed in this research cover only a small portion of the procedures that computational scientists need for their workflows. Eventually, we wish to allow for the development of a large collection of generic workflow components. We can take steps in this direction by building on our existing components. For example, *Magnitude* performs a relatively simple operation on multi-dimensional data, where one dimension spans a number of quantities involved in each instance of the operation. This model can fit any number of operations involving a repeating, fixed number of quantities, and it can even be made to work with a formula specified by the user at runtime. This opens the door to a large family of generic components.

Acknowledgments

The authors would like to thank Matthieu Dreher and Tom Peterka from Argonne National Lab for their valuable assistance with GROMACS and Pegasus.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

This work was supported by Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357, program manager Lucy Nowell.

References

[1] B. Ludäscher, I. Altintas *et al.*, "Scientific workflow management and the kepler system: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 10, pp. 1039–1065, 2006.

[2] G. Malewicz, I. Foster *et al.*, "A tool for prioritizing DAGMan jobs and its evaluation," *High Performance Distributed Computing, 2006 15th IEEE International Symposium on*, pp. 156–168, 0-0 2006.

[3] F. Zheng, H. Abbasi *et al.*, "PreDatA - preparatory data analytics on Peta-Scale machines," in *In Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium, April, Atlanta, Georgia*, 2010.

[4] C. Docan, M. Parashar, and S. Klasky, "DataSpaces: An interaction and coordination framework for coupled simulation workflows," *HPDC '10: Proceedings of the 18th international symposium on High performance distributed computing*, 2010.

[5] M. Wolf, Z. Cai *et al.*, "Smartpointers: Personalized scientific data portals in your hand," in *Supercomputing, ACM/IEEE 2002 Conference*, Nov 2002, pp. 20–20.

[6] H. Abbasi, M. Wolf *et al.*, "Datastager: scalable data staging services for petascale applications," in *HPDC*, D. Kranzlmüller, A. Bode *et al.*, Eds. ACM, 2009, pp. 39–48.

[7] A. Nisar, W.-k. Liao, and A. Choudhary, "Scaling parallel I/O performance through I/O delegate and caching system," in *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–12.

[8] C. C. Ober, R. A. Oldfield *et al.*, "Seismic imaging on the Intel Paragon," *Computers & Mathematics with Applications*, vol. 35, no. 7, pp. 65 – 72, 1998, advanced Computing on Intel Architectures. [Online]. Available: http://www.sciencedirect.com/science/article/B6TYJ-3SYXG2V-7/2/ce76e3a4ad1f2d8d17e1bdc0751ce513

[9] J. Dayal, D. Bratcher *et al.*, "Flexpath: Type-Based Publish/Subscribe System for Large-scale Science Analytics," in *Cluster, Cloud, and Grid*, ser. CCGrid '14. IEEE, 2014.

[10] J. Dayal, J. Lofstead *et al.*, "Soda: Science-driven orchestration of data analytics," in *e-Science (e-Science), 2015 IEEE 11th International Conference on*, Aug 2015, pp. 475–484.

[11] J. Lofstead, J. Dayal *et al.*, "D2t: Doubly distributed transactions for high performance and distributed computing," in *2012 IEEE International Conference on Cluster Computing*, Sept 2012, pp. 90–98.

[12] ——, "Efficient transactions for parallel data movement," in *Proceedings of the 8th Parallel Data Storage Workshop*, ser. PDSW '13. New York, NY, USA: ACM, 2013, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/2538542.2538567

[13] J. Lofstead and J. Dayal, "Transactional parallel metadata services for application workdflows," in *In Proceedings of High Performance Computing Meets Databases at Supercomputing*, 2012.

[14] J. Lofstead, J. Dayal *et al.*, "Efficient, failure resilient transactions for parallel and distributed computing," in *Proceedings of the 2014 International Workshop on Data Intensive Scalable Computing Systems*, ser. DISCS '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 17–24. [Online]. Available: http://dx.doi.org/10.1109/DISCS.2014.13

[15] J. Lofstead, A. Champsaur *et al.*, "Superglue: Standardizing glue components for hpc workflows," in *IEEE Cluster 2016*, Taipei, Taiwan, September 2016.

[16] J. Lofstead, F. Zheng *et al.*, "Adaptable, metadata rich IO methods for portable high performance IO," in *Proceedings of the International Parallel and Distributed Processing Symposium*, Rome, Italy, 2009.

[17] S. Plimpton, R. Pollock, and M. Stevens, "Particle-mesh ewald and rrespa for parallel molecular dynamics simulations," in *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, PPSC 1997, March 14-17, 1997, Hyatt Regency Minneapolis on Nicollet Mall Hotel, Minneapolis, Minnesota, USA*. SIAM, 1997.

[18] Z. Lin, T. S. Hahm *et al.*, "Turbulent transport reduction by zonal flows: Massively parallel simulations," *Science*, vol. 281, no. 5384, pp. 1835–1837, September 1998.

[19] B. Hess, C. Kutzner *et al.*, "Gromacs 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation," *Journal of chemical theory and computation*, vol. 4, no. 3, pp. 435–447, 2008.

[20] McKesson, "Formfast," 2016, http://formfast.com/platform/integrate/ehr-integration/mckesson/.

[21] wfmc, "Workflow management coalition," 2016, http://wwww.wfmc.org/.

[22] S. J. Mullender, I. M. Leslie, and D. McAuley, "Operating-system support for distributed multimedia," in *Proceedings of the 1994 Summer USENIX Technical Conference*, 1994, pp. 209–219.

[23] M. Wilde, M. Hategan *et al.*, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37, no. 9, pp. 633–652, 2011.

[24] H. Karimabadi, B. Loring *et al.*, "In-situ visualization for global hybrid simulations," in *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery*. ACM, 2013, p. 57.

[25] K. Moreland, D. Lepage *et al.*, "Remote rendering for ultrascale data," *Journal of Physics: Conference Series*, vol. 125, no. 1, p. 012096, 2008. [Online]. Available: http://stacks.iop.org/1742-6596/125/i=1/a=012096

[26] B. Whitlock, J. M. Favre, and J. S. Meredith, "Parallel in situ coupling of simulation with a fully featured visualization system," in *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization*, ser. EGPGV '11. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2011, pp. 101–109. [Online]. Available: http://dx.doi.org/10.2312/EGPGV/EGPGV11/101-109

[27] M. Riedel, T. Eickermann *et al.*, "Computational steering and online visualization of scientific applications on large-scale hpc systems within e-science infrastructures," in *e-Science and Grid Computing, IEEE International Conference on*, dec. 2007, pp. 483 –490.

[28] M. Dreher and T. Peterka, "Bredala: Semantic data redistribution for in situ applications," in *Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. Ieee, 2016.

[29] G. Eisenhauer, M. Wolf *et al.*, "A type system for high performance communication and computation," in *e-Science Workshops (eScienceW), 2011 IEEE Seventh International Conference on*. IEEE, 2011, pp. 183–190.

[30] Yassine Mrabet, "Simple torus," https://commons.wikimedia.org/wiki/File:Simple_Torus.svg, 2007, accessed: April 27, 2016.

[31] A. Champsaur, "Superglue code," https://acham_@bitbucket.org/acham_/superglue.git, 2016.